Using clang as a Frontend on a Formal Verification Tool

Mikhail R. Gadelha[1], Jeremy Morse[2], Lucas Cordeiro[3], Denis A. Nicole[1]
[1]University of Southampton, [2]University of Bristol, [3]Federal University of Amazonas

ESBMC

The Efficient SMT-Based Context-Bounded Model Checker (ESBMC) is an open source, permissively licensed (apache 2), cross platform, *bounded model checker* (BMC) for C/C++ programs. It is written primarily in portable C++ and, using Autotools, builds on multiple platforms. The tool was developed for bounded model checking of both sequential and concurrent programs using a variety of SMT solvers, and has a proven track record of bug finding in real-world applications.

In addition to the fixed unrolling of a conventional BMC, the tool also implements a *k-induction* algorithm to provide proofs of correctness for some unbounded programs. Recent versions of ESBMC offer efficient interfaces to a wide variety of SMT solvers, including Z3, Boolector, MathSAT, Yices and CVC4.

Earlier releases of ESBMC used a modified C parser written by James Roskind and a C++ parser based on OpenC++ as front-ends, together spanning more than 62KLOC; maintaining these was a substantial task, and reduced the amount of effort that can be spent addressing research questions.

ESBMC now uses clang as it's front-end. It was developed using libTooling and it’s around 11KLOC, a fraction of the original front-ends.

Features

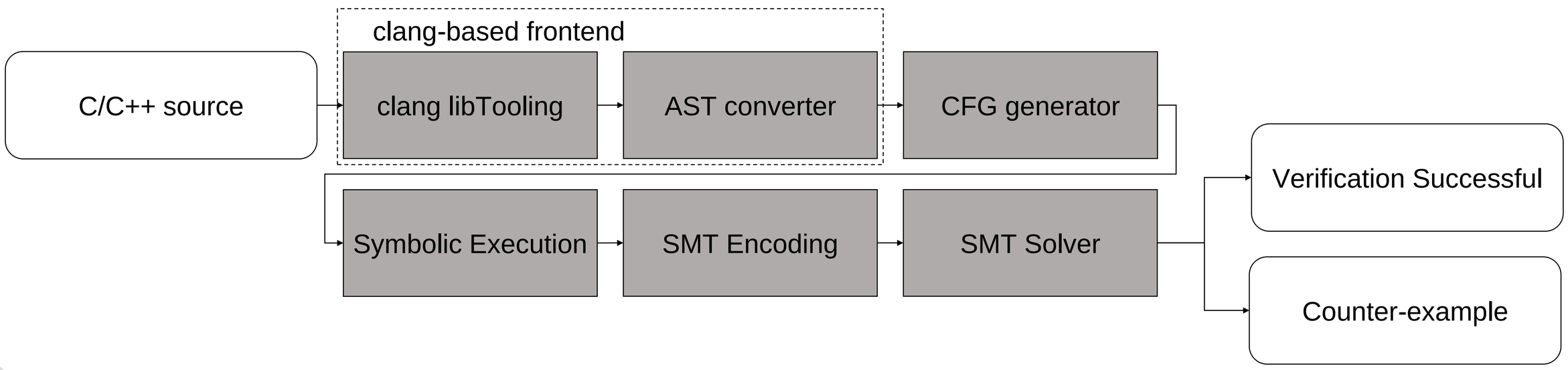
ESBMC encodes a sequential program into a single SMT formula while, when verifying multi-threaded programs, ESBMC adopts an explicit state approach by constructing all necessary interleavings of thread execution, and building an SMT expression for each one of them.

By default, ESBMC checks for violations of the programming language standard. These checks include: bounds (array out of-bound access), division by zero, pointers (NULL dereference, out-of-bounds, or invalid), memory leak, arithmetic over- and under-flow, deadlock, data race, lock order (mutex lock acquisition ordering), and atomicity.

Of course, a program may be correct C, but may still do the wrong thing. It is therefore possible to include additional information in the form of C asserts. These too are coded into appropriate SMT constraints.

ESBMC also offers two encodings when verifying programs with floating-point arithmetic: a fixed-point “fraction” encoding, and a fully IEEE-754 compliant encoding. Currently, only two solvers support the SMT floating-point semantics (MathSAT and Z3), so ESBMC converts all floating-point operations into bit-vector operations when using the other solvers (Boolector, CVC4 and Yices).

Verification process



Why we moved to clang

Our old frontends was developed more than 15 years ago:

- No support for compound literals
- No support for designated initializers
- No support for the typeof operator
- Partial support for C++98
- Bugs and hacks everywhere
- Template instantiation is hard (e.g., C++11 standard, § 14.7.3.7)

Clang provides a well-defined AST and our clang-based frontend is a converter, that reads clang’s AST and converts it into a format understandable by our tool:

- New feature? No problem! We just need to add a new conversion node
- No need to ever program in flex/bison again
- Static expressions can be evaluated by clang for us: sizeof/alignof expressions, static asserts, if a dynamic cast is always null, EvaluateAsInt, EvaluateAsBooleanCondition, EvaluateAsFloat, etc

ESBMC now prints warning and errors as expected from a compiler.

The frontends are now much smaller (from 62KLOC to 11KLOC) and easier to maintain.

Limitations and bug fixing

During the implementation of the frontend, we found the following limitations:

- Random crashes (e.g., broken USR generation in #line directives)
- Clang doesn’t build the Vtable using the defined AST
- No access to the static analyzer
- Lack of documentation in corner cases

We were able to circumvent some of these limitations but still some persist. We submitted the following patches to address two problems:

- **D42966**: Fix USR generation in the presence of #line directives or linemarkes.
- **D36610**: Add option to getFullyQualifiedName using a custom PrintingPolicy.

PLEASE, REVIEW OUR PATCHES! ☺

For further information, publications, and downloads, see:

`http://www.esbmc.org/`

ESBMC is a collaboration between the University of Southampton, UK, the University of Bristol, UK, the Federal University of Amazonas, Brazil, and the Stellenbosch University, South Africa.

Mikhail R. Gadelha
Email: myrg1g14@soton.ac.uk / mikhail.ramalho@gmail.com